FASTLANE: Software Transactional Memory
Optimized for Low Numbers of Threads

Jons-Tobias Wamhoff
Christof Fetzer
Technische Universität Dresden, Germany
first.last@tu-dresden.de

Etienne Rivièere
Pascal Felber
Université de Neuchâtel, Switzerland
first.last@unine.ch

Gilles Muller
INRIA, France
first.last@inria.fr

Software transactional memory (STM) can lead to scalable implementations of concurrent programs, as the relative performance of an application increases with the number of threads that support it. However, the absolute performance is typically impaired by the overheads of transaction management and instrumented accesses to shared memory. This often leads a STM-based program with a low thread count to perform worse than a sequential, non-instrumented version of the same application.

We propose FASTLANE [WFF+13], a new STM system that bridges the performance gap between sequential execution and classical STM algorithms when running on few cores (see Figure 1). FASTLANE seeks to reduce instrumentation costs and thus performance degradation in its target operation range. We introduce a family of algorithms that differentiate between two types of threads: One thread (the master) is allowed to commit transactions without aborting, thus with minimal instrumentation and management costs and at nearly sequential speed, while other threads (the helpers) execute speculatively. Helpers typically run slower than STM threads, as they should contribute to the application progress without impairing on the performance of the master (in particular, helpers never cause aborts for the master’s transactions) in addition to performing the extra bookkeeping associated with memory accesses.

FASTLANE is implemented within a state-of-the-art STM runtime and compiler. Multiple code paths are generated for execution (see Figure 2): sequential on a single core, FASTLANE (master and helper) for few cores, and STM for many cores. Applications can

Figure 1: Bridging the gap between sequential and STM performance for few threads.

Figure 2: Code path for transaction selected dynamically at start by runtime system.
dynamically select a variant at runtime, depending on the number of cores available for execution.

FASTLANE almost systematically wins over a classical STM in the 2-4 threads range, and often performs better than sequential execution of the non-instrumented version of the same application. Figure 3 shows our results for the STAMP Vacation benchmark, an online travel reservation system. We compare against TML, which is based on a global versioned readers-writer lock, NOREC, which extends TML with buffered updates and value-based validation, and TINYSTM, an implementation of the lazy snapshot algorithm with time-based validation using an array of versioned locks.

![Figure 3: Comparison against different STM implementation with the STAMP Vacation benchmark.](image)

The overheads compared to the single master thread origin from saving the context at transaction start and validating all reads during the execution of the transaction (for TML, NOREC and TINYSTM). Fully optimistic STM implementation additionally suffer from overheads due to maintaining a read-set and write-set, which must be validated for conflicts with other threads at commit time, and tracking dynamically managed memory (for NOREC and TINYSTM). The FASTLANE master must only acquire a global lock and reflect all its updates in the meta-data and achieves a performance close to the sequential uninstrumented execution.

The scalability for few threads is achieved by activating FASTLANE’s speculative helper threads that maintain a read-set and write-set. Figure 3 shows their increasing share of the total number of commits the more threads are enabled. TML does not scale because all threads abort and wait as long as a single update transaction is active.

References